Chapitre 3 :

Système de gestion de fichiers

3.1- Introduction

Le système de gestion de fichiers (SGF) est la partie la plus visible d'un système d'exploitation qui se charge de gérer le stockage et la manipulation de fichiers (sur une unité de stockage : partition, disque, CD, disquette. Un SGF a pour principal rôle de gérer les fichiers et d'offrir les primitives pour manipuler ces fichiers.

3.2- Formatage et Partitionnement

A- Partitionnement : consiste à « cloisonner » le disque. Il permet la cohabitation de plusieurs systèmes d'exploitation sur le même disque (il permet d'isoler certaines parties du système). L'information sur le partitionnement d'un disque est stockée dans son premier secteur (secteur zéro), le MBR (Master Boot Record).
Deux types de partitionnement :

- Primaire : On peut créer jusqu'à 4 partitions primaires sur un même disque.
- Etendue est un moyen de diviser une partition primaire en sous-partitions (une ou plusieurs **partitions logiques** qui se comportent comme les partitions primaires, mais sont créées différemment (pas de secteurs de démarrage))

Dans un même disque, on peut avoir un ensemble de partitions (multi-partition), contenant chacune un système de fichier (par exemple DOS et UNIX)

Figure 3. 1. Multipartition d'un disque

B- Formatage : Avant qu'un système de fichiers puisse créer et gérer des fichiers sur une unité de stockage, son unité doit être **formatée** selon les spécificités du système de fichiers. Le formatage inspecte les secteurs, efface les données et crée le répertoire racine du système de fichiers. Il crée également un **superbloc** pour stocker les informations nécessaires à assurer l'intégrité du système de fichiers.

Figure 3. 2. Organisation du système de fichier

Un superbloc contient notamment : L'identifiant du système de fichiers (C:, D : ..), Le nombre de blocs dans le système de fichiers, La liste des blocs libres, l'emplacement du répertoire racine, la date et l'heure de la dernière modification du système de fichiers, une information indiquant s'il faut tester l'intégrité du système de fichiers.

3.3- Le concept de fichier

Un fichier est l'unité de stockage logique mise à la disposition des utilisateurs pour l'enregistrement de leurs données : c'est l'unité d'allocation. Le SE établi la correspondance entre le fichier et le système binaire utilisé lors du stockage de manière transparente pour les utilisateurs. Dans un fichier on peut écrire du texte, des images, des calculs, des programmes…
Les fichiers sont généralement créés par les utilisateurs. Toutefois certains fichiers sont générés par les systèmes ou certains outils tels que les compilateurs.
Afin de différencier les fichiers entre eux, chaque fichier a un ensemble d'attributs qui le décrivent. Parmi ceux-ci on retrouve : le nom, l'extension, la date et l'heur de sa création ou de sa dernière modification, la taille, la protection. Certains de ces attributs sont indiqués par l'utilisateur, d'autres sont complétés par le système d'exploitation.

3.4- La notion de répertoire

Un répertoire est une entité crée pour l'organisation des fichiers. En effet on peut enregistrer des milliers, voir des millions de fichiers sur un disque dur et il devient alors impossible de s'y retrouver. Avec la multitude de fichiers créés, le système d'exploitation a besoin d'une organisation afin de structurer ces fichiers et de pouvoir y accéder rapidement. Cette organisation est réalisée au moyen de **répertoires** également appelés **catalogues** ou **directory**.
Un répertoire est lui-même un fichier puisqu'il est stocké sur le disque et est destiné à contenir des fichiers. Du point de vue SGF, un répertoire est un fichier qui dispose d'une structure logique : il est considéré comme un tableau qui contient une entrée par fichier. L'entrée du répertoire permet d'associer au nom du fichier (nom externe au SGF) les informations stockées en interne par le SGF. Chaque entrée peut contenir des informations sur le fichier (attributs du fichier) ou faire référence à (pointer sur) des structures qui contiennent ces informations.

Dans ce cas, chaque fichier à un i-noeud

On distingue plusieurs structures pour les répertoires :

- La structure **plate à un niveau** : organisée en plusieurs répertoires mais chacun d'eux ne peut contenir que des fichiers. Aujourd'hui absurde, cette approche existait à l'époque des premiers systèmes d'exploitation car le nombre de fichiers était limité.
- La structure à **deux niveaux** : chaque utilisateur dispose de son propre répertoire dans lequel il peut conserver des fichiers et des répertoires.
- La structure **arborescente** : contient un nombre arbitraire de niveaux et chaque répertoire peut contenir des fichiers et des sous répertoires.

Le nom complet d'un fichier est formé d'une liste des répertoires qu'il faut traverser à partir du haut de la hiérarchie (le *répertoire racine* (root directory)) plus le nom_du_fichier. Les répertoires sont séparés par un caractère qui dépend du système d'exploitation : " **>**" pour Multics, "**/**" pour UNIX, " ****" pour Dos et Winxx et " **:** " pour MacOS.

Un tel chemin (exprimé à partir de la racine) est appelé chemin **absolu**. Voici un exemple de chemin absolu sous MS-DOS c:\cours\chapitre4.txt et sous Unix /home/user1/rapport.txt Par contre, un chemin qui ne commence pas par la racine est un chemin **relatif.**

Ces deux concepts de fichier et de répertoire sont considérés par le système d'exploitation comme une seule entité différentiable par un bit à rajouter aux attributs.

3.5- Rôles d'un système de gestion de fichiers

Un SGF a pour principal rôle de gérer les fichiers et d'offrir les primitives pour manipuler ces fichiers. Il effectue généralement les tâches suivantes :

- Fournit une interface conviviale pour manipuler les fichiers (vue fournie à l'utilisateur). Il s'agit de simplifier la gestion des fichiers pour l'utilisateur (généralement, l'utilisateur fournis seulement les attributs nom et extension du fichier, les autres attributs sont gérés implicitement par le SGF). Cette interface fournit la possibilité d'effectuer plusieurs opérations sur les fichiers. Ces opérations permettent généralement d'ouvrir, de fermer, de copier, de renommer des fichiers et des répertoires.
- La gestion de l'organisation des fichiers sur le disque (allocation de l'espace disque aux fichiers)
- La gestion de l'espace libre sur le disque dur
- La gestion des fichiers dans un environnement Multi-Utilisateurs, la donnée d'utilitaires pour le diagnostic, la récupération en cas d'erreurs, l'organisation des fichiers.

3.4.1- La gestion de l'organisation de l'espace disque

Sur le disque, un fichier est sauvegardé sur un ensemble de clusters, appelés également blocs. Le SGF manipule alors des blocs numérotés de 0 à N-1 (N = taille du disque/taille d'un bloc). Chaque fichier (ordinaire ou répertoire) d'un système de fichiers est stocké sur l'unité de stockage du système de fichiers. Ses données sont dans des blocs de taille fixe (512, 1024, ou 2048 octets, ...) et à chaque fichier est alloué un nombre de blocs.
La lecture ou l'écriture d'un élément d'un fichier impliquera le transfert vers la mémoire du bloc entier qui contient cet élément.

3.4.1.1- Techniques d'allocation des blocs sur le disque

On distingue trois manières d'organiser les blocs d'un fichier : contiguë, chaînée et indexée.

3.4.1.1.1- Allocation contiguë

Pour chaque fichier à enregistrer, le système recherche une zone suffisamment grande pour accueillir le fichier. Le fichier sera alors constitué de plusieurs blocs contigus.
Cette méthode présente l'avantage de la rapidité de l'accès (les blocs étant contigus, on limite les déplacements de la tête le lecture/écriture, coûteux en temps). Cependant, elle présente un grand nombre d'inconvénients :

- Le dernier bloc a toutes chances d'être sous-utilisé et ainsi, **on gaspille de la place**. Le pourcentage de place perdue est d'autant plus grand que la taille moyenne des fichiers est faible, ce qui est la réalité
- Il est difficile de prévoir la taille qu'il faut réserver au fichier : un fichier est amené à augmenter de taille, par conséquent il faut prévoir de l'espace libre après le dernier secteur alloué. Si le fichier est agrandi, il faudra le **déplacer** pour trouver un nouvel ensemble de blocs consécutifs de taille suffisante.
- La perte d'espace sur le disque : si on prévoit trop d'espace libre, le fichier risque de ne pas l'utiliser en entier. En revanche, si on prévoit trop peu d'espace libre, le fichier risque de ne pas pouvoir être étendu.
- Problème de **fragmentation externe** : c'est l'espace perdu en dehors des fichiers. On peut effacer des données ou supprimer des fichiers ce qui libère des blocs sur le disque. Au fil de l'utilisation, il peut se créer un grand nombre de petites zones dont la taille ne suffit souvent pas pour allouer un fichier mais dont

le total correspond a un espace assez volumineux.

Table

3.4.1.1.2- Allocation chaînée (non contiguë)

Le principe est d'allouer des blocs chaînés entre eux aux fichiers. Un fichier peut désormais être éparpillé sur le disque puisque chaque bloc permet de retrouver le bloc suivant. Lorsque le fichier change de taille, la gestion des blocs occupés est simple. Il n'y a donc aucune limitation de taille, si ce n'est l'espace disque lui- même.

Figure 3. 6. Allocation chaînée

Cette méthode présente l'avantage de l'élimination du problème de fragmentation externe. Aussi le faite de ne pas nécessiter une structure spéciale pour sa mise en place, constitue un autre avantage. En revanche, les inconvénients ici aussi sont multiples :

- L'accès au fichier est totalement séquentiel, on doit toujours commencer le parcours du fichier à partir du début.
- La perte d'un chaînage entraîne la perte de tout le reste du fichier. Pire encore, il suffit qu'une valeur soit modifiée dans un pointeur pour qu'on se retrouve dans une autre zone de la mémoire.

3.4.1.1.3- Allocation non contiguë indexée

Tous les inconvénients de l'allocation chaînée peuvent être résolus d'une manière simple : il suffit de retirer les pointeurs des blocs et de les placer dans une structure de données gardée en mémoire centrale, ainsi, les informations sur les numéros de blocs peuvent être obtenue à tout moment.

Figure 3. 7. Allocation indexée

La plus part des systèmes actuels appliquent ce mode. MS-DOS utilise la **FAT** (File Allocation Table) pour y conserver les chaînages entre les blocs. Windows NT utilise la **MFT** (Master File Table) associé au système **NTFS** (New Technology File System) .UNIX, GNU/Linux utilisent le **I-Node** (Index node).

a. FAT

On parle généralement de système de fichiers **FAT16** et **FAT32.**

- Le FAT16 est utilisé par MS-DOS. En FAT16, les numéros de blocs sont écrits sur 16 bits. Si on suppose que la taille d'un bloc est 32Ko, la taille maximale adressables est alors 2Go (2^{16} x 32 Ko = 2097152 Ko = 2Go)
- Le FAT32 est pris en charge par Windows 95 et les versions qui ont suivis. Les numéros de blocs sont écrits sur 32 bits (en réalité, sur 28bits, 4 bits étant réservés). Si on suppose que la taille d'un bloc est de 32 ko, la taille maximale adressable théoriquement est de 8 To (2^{28} x 32 Ko = 8 To). Toutefois, Microsoft la limite volontairement à 32 Go sur les systèmes Windows 9x afin de favoriser NTFS.

b. NTFS

Le système de fichiers NTFS (New Technology File System) est utilisé par Windows2000, WindowsNT, Windows XP et Windows Vista. Il utilise un système basé sur une structure appelée MFT (Master File Table), permettant de contenir des informations détaillées sur les fichiers. Ce système permet ainsi l'utilisation de noms longs, mais, contrairement au système FAT32, il est sensible à la casse, c'est-à-dire qu'il est capable de différencier des noms en majuscules de noms en minuscules.

Figure 3. 8. Partition NTFS

- Coté performances, l'accès aux fichiers sur une partition NTFS est plus rapide que sur une partition de type FAT car il utilise un arbre binaire performant pour localiser les fichiers. La limite théorique de la taille d'une partition est de 16 hexa octets (17 milliards de To), mais la limite physique d'un disque est de 2To (va encoder en 64 bits = 2^{64} = 18 446 744 073 709 551 616 = 16 EiB (1 exbibyte = 1EiB = 2^{60} bytes).

C'est au niveau de la sécurité que NTFS prend toute son importance, car il permet de définir des attributs pour chaque fichier.

c. Structure d'un I-Node

La structure d'I-Node est utilisée par le système de gestion de fichier **ext3fs** d'Unix ou GNU/Linux (ext3fs pour *third extented file system*). Un nœud d'index est constitué d'attributs décrivant le fichier ou le répertoire et d'adresses de blocs contenant des données. Cette structure possède plusieurs entrées, elle permet au système de disposer d'un certain nombre de données sur le fichier :

- la taille,
- l'identité du propriétaire et du groupe : un fichier en Unix est crée par un propriétaire, qui appartient à un groupe,
- Les droits d'accès : pour chaque fichier, Unix définit trois droits d'accès (lecture (r), écriture (w) et exécution (x)) pour chaque classe d'utilisateurs (trois types d'utilisateur {propriétaire, membre du même groupe que le propriétaire, autres}). Donc à chaque fichier, Unix associe neuf droits,
- les dates de création, de dernière consultation et de dernière modification,
- le nombre de références existant pour ce fichier dans le système,
- les dix premiers blocs de données,
- d'autres entrées contiennent l'adresse d'autres blocs (on parle alors de bloc d'indirection) :
  - une entrée pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc de données (simple indirection)
  - Une entrée pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc de données (double indirection)
  - Une entrée pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc de données (triple indirection)

La structure d'I-Node est conçue afin d'alléger le répertoire et d'en éliminer les attributs du fichier ainsi que les informations sur l'emplacement des données.
Une entrée dans un I-Node d'un répertoire contiendra donc un nom d'un fichier ou

sous-répertoire et l'I- Node associé.

Exemple 3. 3

Si on suppose que la taille d'un bloc est de 1Ko, un fichier sous Unix peut avoir la taille maximale suivante : 10 x 1Ko + 256 x 1Ko + 256 x 256 x 1Ko + 256 x 256 x 256 x 1Ko, ce qui donne en théorie

plus de 16Go. En réalité, la taille réelle maximale d'un fichier est inférieure à cette valeur à cause de l'utilisation de pointeurs signés pour le déplacement au sein d'un fichier.
Pour les fichiers les plus longs, trois accès au disque suffisent pour connaître l'adresse de tout octet du fichier.

Figure 3. 9. Exemple d'utilisation d'I-Node

3.4.1.2- La création de fichier par le SE

Un fichier ou un répertoire sont tout deux créé e suivant les mêmes étapes qui sont les suivantes :

- La création d'une structure de données pour décrire le fichier. Les attributs du fichier sont sauvegardés dans cette structure de données.
- La création du fichier proprement dit. Il s'agit d'allouer au fichier un certain nombre de blocs sur le disque selon sa taille. Les blocs d'un fichier vont contenir des données sans aucune structure particulière. Les blocs d'un répertoire ont par contre une structure bien particulière, en effet ils contiennent des noms et des attributs de fichiers et de sous répertoires.

3.4.2- La gestion de l'espace libre sur le disque

Les systèmes d'exploitation utilisent essentiellement deux approches pour mémoriser l'espace libre : une statique et une dynamique.

- **Bitmap :** Approche statique utilise une *table de bits (*vecteur de bits n blocs*)* comportant autant de bits que de blocs sur le disque. A chaque bloc du disque, correspond un bit dans la table, positionné à 1 si le bloc est occupé, à 0 si le bloc est libre (ou vice versa).

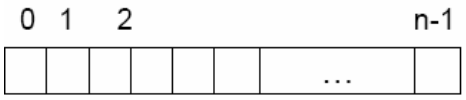

Figure 3. 10. Vecteur de bits à n bloc

Si les blocs 3, 4, 5, 9, 10, 15, 16 sont libres : 11100011100111100…

Cette solution est utilisée pour trouver n blocs contigus, elle est utilisée dans les systèmes : NTFS, ext2fs

Exemple 3. 4

Par exemple, un disque de 300 Mo, organisé en blocs de 1 Ko, sera géré par une table de 300 Kbits qui occupera 38 des 307.200 (300x1024) blocs.

- **Liste chaînée :** Approche dynamique utilise une liste chaînée constituée d'éléments, chacun mémorisant des numéros de blocs libres. Tous les blocs libres sont liés ensemble par des pointeurs.

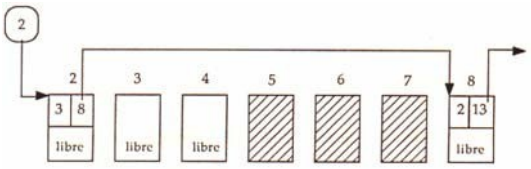

Figure 3. 11. Exemple d'utilisation d'une liste chaînée pour la gestion de l'espace libre

Exemple 3. 5

Par exemple, un disque de 300 Mo, organisé en blocs de 1 Ko. Supposons que chaque bloc soit adressé par 4 octets. Chaque bloc de la liste pourra contenir 255 (1024/4) adresses de blocs libres. La liste comprendra donc au plus 307.200/255 = 1205 blocs. Cette solution mobilise beaucoup plus de place que la précédente.

3.6 Etude de cas : Systèmes de fichiers LINUX

Sous linux, tout est fichier, organisé suivant une unique arborescence (dont la racine est nommée / et dont l'administrateur est root)

3.6.1 Les différentes catégories de fichiers

- **fichiers normaux (-) :** fichiers normaux : **** texte*** : courrier, sources des programmes, scripts, configuration ; **** exécutables*** : programmes en code binaire
  La commande `ls –l` donne :

  ```
  -rwxrw-r-- 1 etudiant 2LR 34568 avril 3 14 :34 mon-fichier
  ```

- **fichiers répertoires (*d*) :** ce sont des fichiers conteneurs qui contiennent des références à d'autres fichiers. ils permettent d'organiser les fichiers par catégories
  La commande `ls –l` sur un répertoire donne

  ```
  drwxr-x--- 1 etudiant 2LR 13242 avril 2 13 :14 mon-répertoire
  ```

- **fichiers spéciaux :** situés dans `/dev`, ce sont les points d'accès préparés par le système aux périphériques. Le montage va réaliser une correspondance de ces fichiers spéciaux vers leur répertoire "point de montage". Par exemple, le fichier `/dev/hda` permet l'accès et le chargement du 1er disque IDE
- **fichiers liens symboliques (*l*)** : ce sont des fichiers qui ne contiennent qu'une référence (un pointeur) à un autre fichier. Cela permet d'utiliser un même fichier sous plusieurs noms sans avoir à le dupliquer sur le disque.
  La commande `ls –l` pour un lien donne

  ```
  lrwxrwxrwx 1 root root 14 Aug 1 01:58 Mail -> ../../bin/mail*
  ```

Remarque : Sous un système UNIX, un fichier quel que soit son type est identifié par un numéro appelé numéro d'inode (i-nœud). Ainsi derrière la façade du shell, un répertoire n'est qu'un fichier, identifié aussi
par un inode, contenant une liste d'inode représentant chacun un fichier.
Pour connaître le numéro d'inode d'un fichier, on utilise la commande

```
$ls -i mon-fichier
```

3.6.1.1 Parcourir et lister les répertoires

Voici les commandes indispensables (suivies bien sûr d'une validation) pour visiter l'arborescenc*e.*

`$ls` *commande générale d'accès aux infos des fichiers du répertoire courant* (`ls`, `ls -l`, `ls –a`)
`$cd [chemin]` *le chemin peut être absolu ou relatif*
`cd ..` *remonter un niveau (vers le répertoire parent)*
`$cd` *raccourci vers le répertoire personnel*
`$pwd` *donne le nom complet du répertoire courant*
`$mkdir rep` *pour créer un sous-repertoire du répertoire courant*
`$rmdir rep` *pour supprimer un sous-répertoire vide*
`$mv répertoire répertoire-d'accueil/` *déplacement d'un répertoire*
`$mv répertoire nouveau-nom` *Changement de nom d'un répertoire*

3.6.1.2 Commandes de gestion des fichiers

Pour gérer les fichiers vous disposez des commandes suivantes:

`$touch mon-fichier` *création d'un fichier vide*,
`$more mon-fichier` *visualisation d'un fichier page à page*,
`$rm mon-fichier` *suppression d'un fichier*,

`$mv mon-fichier répertoire_accueil` déplacement d'un fichier,
`$mv mon-fichier nouveau-nom` *changement de nom d'un fichier*,
`$cp nom-fichier répertoire-accueil/autre-nom` *copie de fichier*,
`$file mon-fichier` *pour savoir si on a un fichier binaire (exécutable) ou un fichier texte. On obtient pour un fichier texte, comme sortie mon-fichier : ascii text*

3.6.1.3 Créer des liens (*ln*)

Les liens sont utiles pour faire apparaître un même fichier dans plusieurs répertoires, ou sous des noms différents. Ils évitent les duplications et assurent la cohérence des mises à jour.
On distingue en fait deux sortes de liens :

1. **Les liens durs** associent deux ou plusieurs fichiers à un même espace sur le disque, les deux fichiers restant indépendants.
   `$ln rapport.txt /home/etudiant/rapport-lien-dur.txt`
   Le fichier `rapport-lien-dur` est créé dans le répertoire `/home/etudiant`. On peut constater que ces 2 fichiers ont la même taille. Au niveau gestion ils sont indépendants, tout en partageant le même espace disque et donc le même inode. Toute modification de l'un, modifie l'autre. Mais la suppression de l'un, casse le lien, mais ne supprime pas physiquement l'autre.
2. **Les liens symboliques**

   `$ln -s rapport.txt /home/etudiant/rapport-lien-s.txt`

   La commande `ls -F` passée dans le répertoire `/home/etudiant` montre que le fichier `rapport-lien-s.txt` pointe sur `rapport.txt` (ainsi, une requête sur `rapport-lien- s.txt`, va ouvrir `rapport.txt`)

   Le lien symbolique fait référence à un fichier dans un répertoire. La suppression du fichier source entraînera un changement de comportement du fichier lien qui sera considéré comme "cassé" ("broken").

Remarque : La différence entre un lien hard et symbolique se trouve au niveau de l'inode, un lien hard n'a pas d'inode propre, il a l'inode du fichier vers lequel il pointe. Par contre un lien symbolique possède sa propre inode.

3.6.2 Monter un système de fichiers

- Comme le système de fichiers Linux se concentre dans une seule arborescence de fichiers, l'accès et l'utilisation de systèmes extérieurs (disques, disquettes, CD..) doit s'effectuer par **intégration** de ces systèmes de fichiers dans le système fondamental *"racine"*.Ce mécanisme d'intégration, souple et paramétrable, s'appelle le **montage**.
- Techniquement, l'opération de montage consiste à mettre en relation :
  - un fichier de périphérique situé dans `/dev` (qui permet la communication physique avec les données du périphérique)
  - avec un noeud d'insertion dans l'arborescence, appelé son **point de montage**
- Naturellement le montage fondamental est celui du répertoire racine. Celui-ci a dû être déclaré (obligatoirement) **après** le partitionnement des disques et **avant** toute installation sur disque !

- Il est fondamental de bien comprendre ce concept : il conditionne tout accès à une ressource externe, en particulier à des ressources réseau à d'autres disques Linux (voir le processus d'exportation NFS chez le serveur, complémentaire du montage chez le client de la ressource)

3.6.2.1 Commandes de montage/démontage

Il est toujours possible de monter "à la main" les systèmes de fichiers stockés sur les périphériques disques, cd ... avec la commande interactive `mount/umount`

- **Syntaxe générale :**

`$mount -t <type > -o options /dev/rep-spécial  /mnt/rep-montage`

Si cette description est présente dans le fichier `/etc/fstab`, la commande peut être simplifiée

`$mount  /dev/rep-spécial ou  mount  /mnt/rep-montage`

- **Les types principaux**

ext2 (type par défaut), vfat, FAT16 ou FAT32 (Win95 ou Win98), nfs (système de fichiers distant situé sur un serveur NFS)

- **Les options**

Les options par défaut sont:
rw (accès complet), **suid** (les éventuels permissions SUID et SGID des fichiers seront pris en compte), **dev** (permettre l'utilisation des fichiers de périphériques, **exec** (permettre l'exécution de fichiers binaires)

Exemple 3. 6

- `Mount` liste tous les systèmes de fichiers actuellement montés
- `mount –a` monter tous les systèmes au démarrage, exécute /etc/rc.d/rc.sysinit
- `mount  /dev/cdrom` monte le système du cd-rom (si décrit dans `fstab`)
- `umount /mnt/floppy` démonte le système de fichiers disquette
- `mount -t vfat -o uid=5001,gid=5000,umask=022 /dev/hda1 /mnt/disk-c`

monter la partition Windows occupant la 1ère partition `/dev/hda1` dans le répertoire `/mnt/disk-c`, avec les options : l'utilisateur d'`uid 5001`, et le groupe de `gid 500`, seront propriétaires de tous les fichiers, la création d'un fichier s'effectuera avec le `umask` 022, c'est-à-dire les permissions 755 `(rwxr-xr-x)`.

3.6.3 Installer une nouvelle partition

Dans certains cas il peut s'avérer indispensable d'étendre le système de fichiers sur un nouveau disque dur, ou une partition récupérée.
L'objectif consiste à assigner à une sous-arborescence du système de fichiers, cette nouvelle ressource périphérique, par le processus de montage.
Soit une nouvelle partition `/dev/hda3`, jusqu'ici "libre", à monter sur `/home`.

1. Avec `fdisk`, lui affecter un système 83 linux
2. La formater : `$mkfs -t ext2 -c -v /dev/hda3`

Formate en blocs de 1024 en vérifiant les blocs (-c), puis écrit la table des inodes.

3. effectuer une copie : `$cp -r /home /root`

Le déplacement de `/home`, dans /root par exemple. En effet /home est présent actuellement sur `hda1`, et il va être ensuite physiquement affecté sur `hda3.`

Les répertoires personnels sont actuellement dans `/root/home`

4. monter la partition `hda3` en `/home : $mount /dev/hda3 /home`

5. récupérer le contenu de /home : $mv /root/home/* /home
6. pour automatiser le montage de /dev/hda3 lors d'un redémarrage du système, ajouter dans la table de montage /etc/fstab la ligne :

/dev/hda3 /home ext2 defaults 1 2

3.6.3.1 Le fichier /etc/fstab

Le processus init (exécuté au démarrage), après chargement du noyau, vérifie les systèmes de fichiers déclarés dans la table du fichier et effectue leur éventuel montage automatique.

Ce fichier /etc/fstab constitue une véritable *"table de montage"*. Il fait l'inventaire des divers systèmes de fichiers que le noyau Linux est susceptible de gérer, précise la façon de les monter, s'ils doivent l'être au démarrage, etc ..

Structure de fstab

Sur chaque ligne on trouve la description du montage d'un système, avec 6 champs :

1. nom du fichier spécial (ou du système distant)
2. nom du point de montage, habituellement un sous-rep (éventuellement à créer) de /mnt
3. le type de fichiers : ext2 (Linux), msdos, vfat (Win9x), ntfs (NT), iso9660 (Cd-rom), nfs
4. liste d'options de montage, séparés par des virgules

   Les options par défaut sont **rw**,**suid**, **dev**, **exec**, **auto**, **nouser**
   - auto/noauto , pour demander/empêcher un montage automatique au démarrage
   - user/nouser, pour autoriser/interdire un user qq (pas le "root") à effectuer le montage
5. paramètre pour dump (commande de sauvegarde) : Une valeur 0 signifie que le système de fichiers ne sera pas sauvegardé lors d'un dump
6. paramètre pour fsck(commande de vérification des fichiers). Il indique l'ordre dans lequel fsck devra vérifier les fichiers, **1** en priorité (c'est normalement la partition racine **/**, **2** sinon, et **0** pour ne pas demander de vérification.

Exemple 3. 7

```
/dev/hda1    /mnt/diskc    vfat    user, auto,rw
```

signifie :

/dev/hda1 est le descripteur de périphérique 1ère partition du 1er disque IDE

/mnt/diskc est le répertoire de montage

vfat est le type de système de fichiers (autres ext2, msdos, iso9660, nfs, swap)

Exemple 3. 8

```
/dev/hdb1    /mnt/disk_d    vfat    user, auto
```

Au lancement du système, ou par la commande mount -a, le système de fichiers Windows 95, installé sur la 1ère partition du 2ème disque (unité D:\), sera monté automatiquement par tous les utilisateurs et accessible dans le répertoire /mnt/disk_d